© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved, worldwide

BASIC-E REFERENCE MANUAL

1/27/77

Micro-Computer Laboratory

Naval Postgraduate School

Monterey, California

Gordon E. Eubanks, Jr.

15 December 1976

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved, worldwide

INTRODUCTION TO BASIC-E

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved, worldwide

BASIC-E is a compiled, scientifically-oriented, fast-executing BASIC language processor. BASIC-E was written by Gordon E. Eubanks, Jr. at the Naval Postgraduate School, Monterey, California, and is in the public domain. Although IMSAI has no current plans to maintain it, we feel this BASIC will be a useful and valuable programming tool. A copy of BASIC-E is being included with each order for IMSAI CP/M version 1.31 revision 1 or newer.

BASIC-E operates on the IMSAI 8080 under the CP/M floppy disc operating system in a minimum of 20K of RAM.

The BASIC-E language features include sequential and random diskette file capabilities, a wide range of predefined functions, user-defined functions, and strings up to 255 characters long. Arrays may have any number of subscripts and may be string or numeric. In a string array, each element can be a string of up to 255 characters. Numbers in BASIC-E are represented internally in binary floating point and approximately 7 significant decimal digits are carried. BASIC-E currently has no provision for formatted printing, and no provision for calling machine language subroutines.

Unlike most microcomputer BASICs, BASIC-E is not interactive. Rather than typing statements into the BASIC language processor, the user prepares a file containing the desired program (usually the editor is used to do this), then compiles it with the BASIC-E compiler, then executes it as many times as he wishes with the BASIC-E run-time interpreter. Since the program is stored in source language form on a disk file, many of the operating system's facilities -- ED, READ, TYPE, LIST, etc. -- may be applied to it. Similarly, data files produced and used by BASIC-E programs are in ASCII form and can be EDited, READ, TYPEd, LISTed, etc.

In the documentation distributed with IMSAI CP/M, Gordon Eubanks' "BASIC-E Reference Manual" follows this introduction; it should be consulted for more detailed information than given in the following summaries.

BASIC-E Language Summary

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved. worldwide

Numbers

Range 2.7E-39 to 3.6E38
Approximately 7 significant decimal digits are represented.

Strings

May contain 0 to 255 ASCII characters. Storage for strings is automatically allocated as required during program execution. In a string array (subscripted string variable) each element is a separate string of characters.

Names (identifiers)

Variable names may be up to 31 characters long. The first character must be a letter; the rest must be letter, digit, or period.

Numeric operators

```
(, ), , *, /, +, -
<, <=, >, >=, <>          (relational)
LT, LE, GT, GE, EQ, NE    (same as preceding line)
NOT, AND, OR
```

String Operators

```
+                    (concatenation)
relational operators as above
```

Statements - General

Line numbers are optional in BASIC-E programs except as required on lines referenced by GOTO, GOSUB, etc. Multiple statements on one line should be seperated by : (except IF, DIM, DEF, and END should be the only statement on the line). A statement may be continued on the next line if a \ precedes the carriage return. Long variable names and remarks may be used freely as they do not effect the size of the program after compilation.

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved, worldwide

Statement Types

```
LET
IF ... THEN line number
IF ... THEN any statement
IF ... THEN ... ELSE ...
FOR
NEXT
GOTO
GOSUB
RETURN
ON ... GOTO list of line numbers
ON ... GOSUB list of line numbers
RANDOMIZE
PRINT
READ
OUT port, data
DATA
RESTORE
REM
END
```

DIM Statements

```
DIM name(dimensions),name(dimensions),...
```

Any number of subscripts are allowed. All arrays must be DIMensioned and the DIM statement should appear in the program before the array is referenced.

```
Examples: DIM N(100)
          DIM BIG(2,4,3,3,2,2,8), DATASTRINGS$(10,5,23)
```

User-Defined Functions

```
DEF FNname(arguments)=expression
```

```
Examples: DEF FNA(X+Y)=X+Y-A
          DEF FNXY.COMBINE$(A$+B$)=A$+", "+B$
```

Sequential File Operations

FILE variable$, variable$ ...

Opens one or more files. Each string variable's value is taken as the name of a file to open; the first file opened is #1, the second, #2, etc. Up to six files can be open at one time.

PRINT #filenumber; list

Outputs a list of variables or expressions to the indicated file. "filenumber" is a numeric expression; note the # before filenumber and the ; after it.

READ #filenumber; list

Reads a list of variables from indicated open file.

CLOSE filenumber, filenumber ...

Closes one or more files.

IF END #filenumber THEN line number

Specifies line number to GOTO if and when a subsequent INPUT attempts to read beyond the end of the indicated file.

Random Access File Operations

FILE variable$ (recordlen), variable$ (recordlen) ...

Opens files for Random access. Recordlen is an expression specifying the length of each record in the file, that is, how many bytes are to be transmitted with each PRINT or INPUT statement.

PRINT #filenumber, recordnumber; list
READ #filenumber, recordnumber; list

These access the specified record of a file. Note that there is a comma before the record number expression and a semicolon after it.

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved, worldwide

Predefined Numeric Functions

X represents any numeric expression

ABS(X)	
ATN(X)	
COS(X)	
COSH(X)	hyperbolic cosine
FRE	number of bytes of unused storage
INP(X)	input from port X
INT(x)	
LOG(X)	
POS(x)	carriage position
RND	random number between 0 and 1
SGN(X)	
SIN(X)	
SINH(X)	hyperbolic sine
SQR(X)	
TAB(X)	
EXP(X)	

Predefined string functions

A$ represents any string expression

ASC(A$)
CHR$(X)
LEFT$(A$)
LEN(A$)
MID$(A$,start,length)
RIGHT$(A$,length)
STR$(X)
VAL(A$)

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved, worldwide

BASIC-E Operation

To use BASIC-E you must have at least 20K of RAM (many applications will require more) and your CP/M operating system must be relocated for the amount of RAM in use as described in section 6.97 of the C/M User's Guide.

BASIC-E source programs reside on diskette files of type BAS. Usually they are input from the terminal with the CP/M editor (ED) or from paper tape with the CP/M READ command. Like any other file, a BASIC-E program may be printed out with the CP/M TYPE or LIST command, copied to another file with PIP, updated with ED, etc.

Before it can be run, a BASIC-E program must be compiled. This is done with the CP/M command

BASIC-E name

where "name" is the name of the file containing the program. The compilation operation lists the program, types error messages if any errors are found in the program, and produces a new file with the same name but of type INT.

The compiled program can then be executed with the command

RUN-E name

The program can be used as many times as desired by reentering the RUN-E command.

To modify the program, EDit the source file (name.BAS), recompile it, then reRUN-E it.

Known Bugs

1. PRINT sometimes makes an error of one in the seventh significant digit; thus, for example, 93 may print out as 92.99999. This error occurs on printout and does not effect arithmetic done in the program.

Notes

1. The commands to compile and run a BASIC-E program are "BASIC-E name" and "RUN-E name", as described in the operation section above, not "BASIC name" and "RUN name" as stated in the BASIC-E Reference Manual.

2. The RANDOMIZE statement is effective only if an INPUT statement has been executed earlier in the program. The variable time it takes the user to respond at the terminal is used to randomize the random number generator.

© 1977 INSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved, worldwide

3. The description of the READ statement on page 44 is rather confused. The three forms of the READ statement are:

(1) [<line number>] READ #<expression>,<expression>;
<variable>{,<variable>}

To read a random record from a specified file.

(2) [<line number>] READ #<expression>;
<variable>{,<variable>}

To read the next sequential record from a specified file.

(3) [<line number>] READ <variable>{,<variable>}

To read values from DATA statements in the program.

4. A CLOSE statement sometimes produces an apparently erronious error message from the compiler. Usually the CLOSE statement can be omitted as all files are automatically closed at the end of program execution.

5. In some cases, when a file is written with a BASIC-E program, garbage is left in the file after the last record. This garbage shows up when the file is later TYPEd, LISTed, or EDited. If you have trouble with this problem try putting an ASCII control-Z character

```
PRINT #n; CHR$(26)
```

where n is the file number.

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved, worldwide

TABLE OF CONTENTS

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved, worldwide

I. INTRODUCTION 3

II. DESCRIPTION OF THE BASIC-E LANGUAGE 5

III. OPERATING INSTRUCTIONS FOR BASIC-E 65

APPENDIX A - COMPILER ERROR MESSAGES 73

APPENDIX B - RUN-TIME MONITOR ERROR MESSAGES 75

APPENDIX C - BASIC-E GRAMMAR 76

APPENDIX D - COMPILER OPTIONS 81

BIBLIOGRAPHY 82

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved, worldwide

I. INTRODUCTION

This manual describes the Naval Postgraduate School BASIC Language (BASIC-E). BASIC-E is an extension of the BASIC language for the 8080 microprocessor. It consists of two subsystems, the compiler and the run-time monitor. The compiler syntax checks a BASIC-E source program producing an object file which may then be executed by the run-time monitor.

BASIC-E is intended to be used in the interactive mode with a CRT or teletype terminal. It includes most features of the proposed ANSI standard BASIC [1] as well as extensive string manipulation and file input/output capabilities. BASIC-E requires a resident disk operating system to handle input/output operations and disk file management. The source program is assumed to reside on the disk.

The BASIC-E compiler consists of a table-driven parser which checks statements for correct syntax and generates code for the BASIC-E machine. The code is executed by the run time monitor. The machine is a zero address stack computer. Floating point arithmetic numbers are represented in 32 bits with an 8 bit exponent, one bit sign and twenty-four bits of fraction. This provides slightly more than seven decimal digits of significance. Variable length strings and n dimensional arrays are both dynamically allocated.

Section II of this manual describes the language elements. Section III provides operating instructions for BASIC-E. The appendices list compiler and run-time error messages, the formal grammar for BASIC-E, and compiler toggles. References are included in the bibliography.

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved, worldwide

II. DESCRIPTION OF THE BASIC-E LANGUAGE

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved. worldwide

Elements of BASIC-E are listed in alphabetical order in this section of the manual. A synopsis of the element is shown, followed by a description and examples of its use. The intent is to provide a reference to the features of this implementation of BASIC and not to teach the BASIC language. References [2,3] provide a good introduction to BASIC programming.

A program consists of one or more properly formed BASIC-E statements. An END statement, if present, terminates the program, and additional statements are ignored. The entire ASCII character set is accepted, but all statements may be written using the common 64-character subset. Section III provides information on source program format.

In this section the "synopsis" presents the general form of the element. Square brackets [] denote an optional feature while braces {} indicate that the enclosed section may be repeated zero or more times. Terms enclosed in < > are either non-terminal elements of the language, which are further defined in this section, or terminal symbols. All special characters and capitalized words are terminal symbols.

ELEMENT:

ABS predefined function

SYNOPSIS:

ABS (<expression>)

DESCRIPTION:

The ABS function returns the absolute value of the <expression>. The argument must evaluate to a floating point number.

EXAMPLES:

```
ABS(X)

ABS(X*Y)
```

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved. worldwide

ELEMENT:

ASC predefined function

SYNOPSIS:

ASC (<expression>)

DESCRIPTION:

The ASC function returns the ASCII numeric value of the first character of the <expression>. The argument must evaluate to a string. If the length of the string is 0 (null string) an error will occur.

EXAMPLES:

```
ASC(A$)

ASC("X")

ASC(RIGHT$(A$,7))
```

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved, worldwide

ELEMENT:

ATN predefined function

SYNOPSIS:

ATN (<expression>)

DESCRIPTION:

The ATN function returns the arctangent of the <expression>. The argument must evaluate to a floating point number.

EXAMPLES:

```
ATN(X)

ATN(SQR(SIN(X)))
```

PROGRAMMING NOTE:

All other inverse trigonometric functions may be computed from the arctangent using simple identities.

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved, worldwide

ELEMENT:

CHR$ predefined function

SYNOPSIS:

CHR$ (<expression>)

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved, worldwide

DESCRIPTION:

The CHR$ function returns a character string of length 1 consisting of the character whose ASCII equivalent is the <expression> converted to an integer modulo 128. The argument must evaluate to a floating point number.

EXAMPLES:

```
CHR$(A)

CHR$(12)

CHR$((A+B/C)*SIN(X))
```

PROGRAMMING NOTE:

CHR$ can be used to send control characters such as a linefeed to the output device. The following statement would accomplish this:

```
PRINT CHR$(10)
```

ELEMENT:

CLOSE statement

SYNOPSIS:

[<line number>] CLOSE <expression> {, <expression>}

© 1977 · IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved, worldwide

DESCRIPTION:

The CLOSE statement causes the file specified by each <expression> to be closed. Before the file may be referenced again it must be reopened using a FILE statement.
An error occurs if the specified file has not previously appeared in a FILE statement.

EXAMPLES:

```
    CLOSE 1

150 CLOSE I, K, L*M-N
```

PROGRAMMING NOTE:

On normal completion of a program all open files are closed. If the program terminates abnormally it is possible that files created by the program will be lost.

ELEMENT:

<constant>

SYNOPSIS:

[<sign>] <integer> . [<integer>] [E <sign> <exp>]

["] <character string> ["]

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved, worldwide

DESCRIPTION:

A <constant> may be either a numeric constant or a string constant. All numeric constants are stored as floating point numbers. Strings may contain any ASCII character.

Numeric constants may be either a signed or unsigned integer, decimal number, or expressed in scientific notation. Numbers up to 31 characters in length are accepted but the floating point representation of the number maintains approximatly seven significant digits (1 part in 16,000,000). The largest magnitude that can be represented is approximately 3.6 times ten to the 38th power. The smallest non-zero magnitude that can be represented is approximately 2.7 times ten to the minus 39th power.

String constants may be up to 255 characters in length. Strings entered from the console, in a data statement, or read from a disk file may be either enclosed in quotation marks or delimited by a comma. Strings used as constants in the program must be enclosed in quotation marks.

EXAMPLES:

```
10

-100.75639E-19

"THIS IS THE ANSWER"
```

ELEMENT:

COS predefined function

SYNOPSIS:

COS(<expression>)

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved, worldwide

DESCRIPTION:

COS is a function which returns the cosine of the <expression>. The argument must evaluate to a floating point number expressed in radians.
A floating point overflow occurs if the absolute value of the <expression> is greater than two raised to the 24th power times pi radians.

EXAMPLES:

```
COS(B)

COS(SQR(X-Y))
```

ELEMENT:

COSH predefined function

SYNOPSIS:

COSH (<expression>)

DESCRIPTION:

COSH is a function which returns the hyperbolic cosine of the <expression>. The argument must evaluate to a floating point number.

EXAMPLES:

```
COSH(X)

COSH(X↑2+Y↑2)
```

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved, worldwide

ELEMENT:

DATA statement

SYNOPSIS:

[<line number>] DATA <constant> {, <constant>}

DESCRIPTION:

DATA statements define string and floating point constants which are assigned to variables using a READ statement. Any number of DATA statements may occur in a program. The constants are stored consecutively in a data area as they appear in the program and are not syntax checked by the compiler.
Strings may be enclosed in quotation marks or optionally delimited by commas.

EXAMPLES:

```
10  DATA 10.0,11.72,100

    DATA "XYZ",11.,THIS IS A STRING
```

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved worldwide

ELEMENT:

DEF statement

SYNOPSIS:

[<line number>] DEF <function name> (<dummy argument list>) = <expression>

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved, worldwide

DESCRIPTION:

The DEF statement specifies a user defined function which returns a value of the same type as the <function name>. One or more <expressions> are passed to the function and used in evaluating the expression. The passed values may be in string or floating point form but must match the type of the corresponding dummy argument. Recursive calls are not permitted. The <expression> in the define statement may reference <variables> other than the dummy arguments, in which case the current value of the <variable> is used in evaluating the <expresssion>. The type of the function must match the type of the <expression>.

EXAMPLES:

```
10  DEF FNA(X,Y) = X + Y - A

    DEF FNB$(A$,B$) = A$ + B$ + C$

    DEF FN.COMPUTE(A,B) = A + B - FNA(A,B) + D
```

ELEMENT:

DIM statement

SYNOPSIS:

[line number] DIM <identifier> (<subscript list>)
{,<identifier> (<subscript list>)}

DESCRIPTION:

The dimension statement dynamically allocates space for floating point or string arrays. String array elements may be of any length up to 255 bytes and change in length dynamically as they assume different values. Initially, all floating point arrays are set to zero and all string arrays are null strings. An array must be dimensioned explicitly; no default options are provided. Arrays are stored in row major order.

Expressions in subscript lists are evaluated as floating point numbers and rounded to the nearest integer when determining the size of the array. All subscripts have an implied lower bound of 0.

When array elements are referenced a check is made to ensure the element resides in the referenced array.

EXAMPLES:

```
DIM A(10,20), B(10)

DIM B$(2,5,10),C(I + 7.3,N),D(I)

DIM X(A(I),M,N)
```

PROGRAMMING NOTE:

A <DIM statement> is an executable statement, and each execution will allocate a new array.

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved worldwide

ELEMENT:

END statement

SYNOPSIS:

[line number] END

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved, worldwide

DESCRIPTION:

An END statement indicates the end of the source program. It is optional and, if present, it terminates reading of the source program. If any statments follow the END statement they are ignored.

EXAMPLES:

```
10  END

    END
```

ELEMENT:

EXP (<expression>)

DESCRIPTION:

The EXP function returns e (2.71828....) raised to the power of the <expression>. The argument must evaluate to a floating point number.
If the value of the <expression> exceeds two to the 127th power, a floating point overflow occurs.

EXAMPLES:

```
EXP(X)

EXP(LOG(X))
```

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved, worldwide

ELEMENT:

<expression>

DESCRIPTION:

Expressions consist of algebraic combinations of variables, constants, and operators. The hierarchy of operators is:

1) ()
2) ↑
3) *, /
4) +, -, concat (+), unary +, unary -
5) relational ops <, <=, >, >=, =, <>
   LT, LE, GT, GE, EQ, NE
6) NOT
7) AND
8) OR, XOR

Relational operators result in a 0 if false and -1 if true. NOT, AND, and OR are performed on 32 bit two's complement binary representation of the integer portion of the variable. The result is then converted to a floating point number. String variables may be operated on by relational operators and concatenation only. Mixed string and numeric operations are not permitted.

EXAMPLES:

```
X + Y

A$ + B$

(A <= B) OR (C$ > D$) / (A - B AND D)
```

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved, worldwide

ELEMENT:

FILE statement

SYNOPSIS:

[<line number>] FILE <variable> [(<expression>)]
{, <variable>[(<expression>)]}

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved, worldwide

DESCRIPTION:

A file statement opens files used by the program. The order of the names determines the numbers used to reference the files in READ and PRINT statements. The value assigned to the first simple variable is file1, the second is file 2, and so forth. There may be any number of FILE statements in a program, but there is a limit to the number of files which may be opened at one time. Currently this limit is set at 6 files. The optional <expression> designates the logical record length of the file. If no length is specified, the file is written as a continuous string of fields with carriage return linefeed characters separating each record. If the record length is present, a carriage return linefeed will be appended to each record. The <variable> must not be subscripted and it must be of type string.

EXAMPLES:

```
FILE INPUT$, OUTPUT$

FILE TABLE.INC$, TAX.INC$(160), PAY.AMT.DAY$(N*3-J)
```

PROGRAMMING NOTE:

The run-time monitor will always assign the lowest available (not previously assigned) number to the file being opened. Thus if files are closed and others opened it is possible that number assignment may vary with program flow.

ELEMENT:

FOR statement

SYNOPSIS:

[<line number>] FOR <index> = <expression> TO
<expression> [STEP <expression>]

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved, worldwide

DESCRIPTION:

Execution of all statements between the FOR statement and its corresponding NEXT statement is repeated until the indexing variable, which is incremented by the STEP <expression> after each iteration, reaches the exit criteria. If the step is positive, the loop exit criteria is that the index exceeds the value of the TO <expression>. If the step is negative, the index must be less than the TO <expression> for the exit criteria to be met.
The <index> must be an unsubscripted variable and is initially set to the value of the first <expression>. Both the TO and STEP expressions are evaluated on each loop, and all variables associated with the FOR statement may change within the loop. If the STEP clause is omitted, a default value of 1 is assumed. A FOR loop is always executed at least once. A step of 0 may be used to loop indefinitely.

EXAMPLES:

```
FOR I = 1 TO 10 STEP 3

FOR INDEX = J*K-L TO 10*SIN(X)

FOR I = 1 TO 2 STEP 0
```

PROGRAMMING NOTE:

If a step of 1 is desired the step clause should be omitted. The execution will be substantially faster since less runtime checks must be made.

ELEMENT:

FRE predefined function

SYNOPSIS:

FRE

DESCRIPTION:

The FRE function returns the number of bytes of unused space in the free storage area.

EXAMPLE:

FRE

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved. worldwide

FUNCTION:

<function name>

SYNOPSIS:

FN<identifier>

© 1977 · IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved, worldwide

DESCRIPTION:

Any <identifier> starting with FN refers to a user-defined function. The <function name> must appear in a DEF statement prior to being used in an <expression>. There may not be any spaces between the FN and the <identifier>.

EXAMPLES:

```
FNA

FN.BIGGER.$
```

ELEMENT:

GOSUB statement

SYNOPSIS:

[<line number>] GOSUB <line number>
[<line number>] GO SUB <line number>

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved, worldwide

DESCRIPTION:

The address of the next sequential instruction is saved on the run-time stack, and control is transferred to the subroutine labeled with the <line number> following the GOSUB or GO SUB.

EXAMPLES:

```
10  GOSUB   300

    GO SUB  100
```

PROGRAMMING NOTE:

The max depth of GOSUB calls allowed is controlled by the size of the run-time stack which is currently set at 12.

ELEMENT:

GOTO statement

SYNOPSIS:

[<line number>] GOTO <line number>

[<line number>] GO TO <line number>

DESCRIPTION:

Execution continues at the statement labeled with the <line number> following the GOTO or GO TO.

EXAMPLES:

```
100 GOTO 50

    GO TO 10
```

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved, worldwide

ELEMENT:

<identifier>

SYNOPSIS:

<letter> { <letter> or <number> or . } [$]

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved, worldwide

DESCRIPTION:

An identifier begins with an alphabetic character followed by any number of alphanumeric characters, or periods. Only the first 31 characters are considered unique. If the last character is a dollar sign the associated variable is of type string, otherwise it is of type floating point.

EXAMPLES:

A

B$

XYZ.ABC

PAY.RECORD.FILE.NUMBER.76

PROGRAMMING NOTE:

All lowercase letters appearing in an <identifier> are converted to uppercase unless compiler toggle D is set to off.

ELEMENT:

IF statement

SYNOPSIS:

[<line number>] IF <expression> THEN <line number>

[<line number>] IF <expression> THEN <statement list>

[<line number>] IF <expression> THEN <statement list>
ELSE <statement list>

DESCRIPTION:

If the value of the <expression> is not 0 the statements which make up the <statement list> are executed. Otherwise the <statement list> following the ELSE is executed, if present, or the next sequential statement is executed.
In the first form of the statement if the <expression> is not equal to 0, an unconditional branch to the label occurs.

EXAMPLES:

```
IF A$ < B$ THEN X= Y*Z

IF (A$<B$) AND (C OR D) THEN 300

IF B THEN X = 3.0 : GOTO 200

IF J AND K THEN GOTO 11 ELSE GOTO 12
```

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved, worldwide

ELEMENT:

IF END statement

SYNOPSIS:

[<line number>] IF END #<expression> THEN <line number>

DESCRIPTION:

If during a read to the file specified by the <expression>, an end of file is detected control is transferred to the statement labeled with the line number following the THEN.

EXAMPLES:

```
        IF END # 1 THEN 100

    10  IF END # FILE.NUMBER - INDEX THEN 700
```

PROGRAMMING NOTE:

On transfer to the line number following the THEN the stack is restored to the state prior to the execution of the READ statement which caused the end of file condition.

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved, worldwide

ELEMENT:

INP predefined function

SYNOPSIS:

INP (<expression>)

DESCRIPTION:

The INP function performs an input operation on the 8080 machine port represented by the value of the <expression> modulo 256 returning the resulting value. The argument must evaluate to a floating point number.

EXAMPLES:

```
INP(2)

INP(CURRENT.INPUT.PORT)
```

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved, worldwide

ELEMENT:

INPUT statement

SYNOPSIS:

```
[<line number>] INPUT [<prompt string> ;]
                      <variable> {, <variable> }
```

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved, worldwide

DESCRIPTION:

The <prompt string>, if present, is printed on the console. A line of input data is read from the console and assigned to the variables as they appear in the variable list. The data items are separated by commas and/or blanks and terminated by a carriage return. Strings may be enclosed in quotation marks. If a string is not enclosed by quotes, the first comma terminates the string. If more data is requested than was entered, or if insufficient data items is entered, a warning is printed on the console and the entire line must be reentered.

EXAMPLES:

```
10  INPUT A,B

    INPUT "SIZE OF ARRAY?"; N

    INPUT "VALUES?"; A(I),B(I),C(A(I))
```

PROGRAMMING NOTE:

Trailing blanks in the <prompt string> are ignored. One blank is always supplied by the system.

ELEMENT:

INT predefined function

SYNOPSIS:

INT (<expression>)

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved, worldwide

DESCRIPTION:

The INT function returns the largest integer less than or equal to the value of the <expression>. The argument must evaluate to a floating point number.

EXAMPLES:

```
INT (AMOUNT / 100)

INT(3 * X * SIN(Y))
```

ELEMENT:

LEFT$ predefined function

SYNOPSIS:

LEFT$ (<expression> , <expression>)

DESCRIPTION:

The LEFT$ function returns the n leftmost characters of the first <expression>, where n is equal to the integer portion of the second <expression>. An error occurs if n is negative. If n is greater than the length of the first <expression> then the entire expression is returned. The first argument must evaluate to a string and the second to a floating point number.

EXAMPLES:

LEFT$ (A$,3)

LEFT$(C$+D$,I-J)

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved. worldwide

ELEMENT:

LEN predefined function

SYNOPSIS:

LEN (<expression>)

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved, worldwide

DESCRIPTION:

The LEN function returns the length of the string <expression> passed as an argument. Zero is returned if the argument is the null string.

EXAMPLES:

```
LEN(A$)

LEN(C$ + B$)

LEN(LASTNAME$ + "," + FIRSTNAME$)
```

ELEMENT:

LET statement

SYNOPSIS:

[<line number>] [LET] <variable> = <expression>

DESCRIPTION:

The <expression> is evaluated and assigned to the <variable> appearing on the left side of the equal sign. The type of the <expression>, either floating point or string, must match the type of the <variable>.

EXAMPLES:

```
100 LET A = B + C

    X(3,A) = 7.32 * Y + X(2,3)

 73 W = (A<B)  OR (C$>D$)

    AMOUNT$ = DOLLARS$ + "." + CENTS$
```

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved worldwide

ELEMENT:

<line number>

SYNOPSIS:

<digit> { <digit> }

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved, worldwide

DESCRIPTION:

<line numbers> are optional on all statements and are ignored by the compiler except when they appear in a GOTO, GOSUB, or ON statement. In these cases, the <line number> must appear as the label of one and only one <statement> in the program.
<line numbers> may contain any number of digits but only the first 31 are considered significant by the compiler.

EXAMPLES:

100

.

4635276353

ELEMENT:

LOG predefined function

SYNOPSIS:

LOG (<expression>)

DESCRIPTION:

The LOG function returns the natural logarithm of the absolute value of the <expression>. The argument must evaluate to a non-zero floating point number.

EXAMPLES:

```
LOG (X)

LOG((A + B)/D)

LOG10 = LOG(X)/LOG(10)
```

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved. worldwide

ELEMENT:

MID$ predefined function

SYSNOPSIS:

MID$ (<expression> , <expression> , <expression>)

DESCRIPTION:

The MID$ function returns a string consisting of the n characters of the first <expression> starting at the mth character. The value of m is equal to the integer portion of the second <expression> while n is the integer portion of the third <expression>.
The first argument must evaluate to a string, and the second and third arguments must be floating point numbers. If m is greater than the length of the first <expression> a null string is returned. If n is greater than the number of characters left in the string all the characters from the mth character are returned. An error occurs if m or n is negative.

EXAMPLES:

```
MID$(A$,I,J)

MID$(B$+C$,START,LENGTH)
```

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved, worldwide

ELEMENT:

NEXT statement

SYNOPSIS:

[<line number>] NEXT [<identifier> {,<identifier>}]

DESCRIPTION:

A NEXT statement denotes the end of the closest unmatched FOR statement. If the optional <identifier> is present it must match the index variable of the FOR statement being terminated. The list of <identifiers> allows matching multiple FOR statements. The <line number> of a NEXT statement may appear in an ON or GOTO statement, in which case execution of the FOR loop continues with the loop variables assuming their current values.

EXAMPLES:

```
10  NEXT

    NEXT I

    NEXT I, J, K
```

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved, worldwide

ELEMENT:

ON statement

SYNOPSIS:

(1) [<line number>] ON <expression> GOTO
<line number> {, <line number>}

(2) [<line number>] ON <expression> GO TO
<line number> {, <line number>}

(3) [<line number>] ON <expression> GOSUB
<line number> {, <line number>}

(4) [<line number>] ON <expression> GO SUB
<line number> {, <line number>}

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved worldwide

DESCRIPTION:

The <expression>, rounded to the nearest integer value, is used to select the <line number> at which execution will continue. If the <expression> evaluates to 1 the first <line number> is selected and so forth. In the case of an ON ... GOSUB statement the address of the next instruction becomes the return address.
An error occurs if the <expression> after rounding is less than one or greater than the number of <line numbers> in the list.

EXAMPLES:

```
10  ON I GOTO 10, 20, 30, 40

    ON J*K-M GO SUB 10, 1, 1, 10
```

ELEMENT:

OUT statement

SYNOPSIS:

[<line number>] OUT <expression> , <expression>

DESCRIPTION:

The low-order eight bits of the integer portion of the second <expression> is sent to the 8080 machine output port selected by the integer portion of the first expression modulo 256. Both arguments must evaluate to floating point numbers.

EXAMPLES:

```
100 OUT 3,10

    OUT PORT.NUM, NEXT.CHAR
```

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved, worldwide

ELEMENT:

POS predefined function

SYNOPSIS:

POS

DESCRIPTION:

The POS function returns the current position of the output line buffer pointer. This value will range from 1 to the print buffer size.

EXAMPLE:

```
PRINT TAB(POS + 3);X
```

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved, worldwide

ELEMENT:

PRINT statement

SYNOPSIS:

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved. worldwide

RANDOM - (1) [<line number>] PRINT #<expression>,<expression>;
<expression> {, <expression> }

SEQUENTIAL - (2) [<line number>] PRINT # <expression> ;
<expression> {, <expression> }

(3) [<line number>] PRINT <expression> <delim>
{ <expression> <delim> }

DESCRIPTION:

A PRINT statement sends the value of the expressions in the expression list to either a disk file (type(1) and (2)) or the console (type (3)). A type (1) PRINT statement sends a random record specified by the second <expression> to the disk file specified by the first <expression>. An error occurs if there is insufficient space in the record for all values.
A type (2) PRINT statement outputs the next sequential record to the file specified by the <expression> following the #.
A type (3) PRINT statement outputs the value of each <expression> to the console. A space is appended to all numeric values and if the numeric item exceeds the right margin then the print buffer is dumped before the item is printed. The <delim> between the <expressions> may be either a comma or a semicolon. The comma causes automatic spacing to the next tab position (14,28,42,56). If the current print position is greater than 56 then the print buffer is printed and the print position is set to zero. A semicolon indicates no spacing between the printed values. If the last <expression> is not followed by a <delim> the print buffer is dumped and the print position set equal to zero. The buffer is automatically printed anytime the print position exceeds 71.

EXAMPLES:

```
100 PRINT #1;A,B,A$+"*"

    PRINT # FILE,WHERE; A/B,D,"END"

    PRINT A, B, "THE ANSWER IS"; x
```

ELEMENT:

RANDOMIZE statement

SYNOPSIS:

[<line number>] RANDOMIZE

DESCRIPTION:

A RANDOMIZE statement initializes the random number generator.

EXAMPLES:

```
10  RANDOMIZE

    RANDOMIZE
```

© 1977 · IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved, worldwide

ELEMENT:

READ statement

SYNOPSIS:

(1) [<line number>] READ <expression>, <expression>;
<variable> {, <variable> }

RANDOM — (2) [<line number>] READ # <expression>;
<variable> {, <variable> }

SEQUENTIAL — (3) [<line number>] READ # <variable> {, <variable> }

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved, worldwide

DESCRIPTION:

A READ statement assigns values to variables in the variable list from either a file (type (2) and (3)) or from a DATA statement (type (1)). Type (2) reads a random record specified by the second expression from the disk file specified by the first expression and assigns the fields in the record to the variables in the variable list. Fields may be floating point or string constants and are delimited by a blank or comma. Strings may optionally be enclosed in quotes. An error occurs if there are more variables than fields in the record.

The type (3) READ statement reads the next sequential record from the file specified by the expression and assigns the fields to variables as described above.

A type (1) READ statement assigns values from DATA statements to the variables in the list. DATA statements are processed sequentially as they appear in the program. An attempt to read past the end of the last data statement produces an error.

EXAMPLES:

```
100 READ A,B,C$

200 READ # 1,I; PAY.REG,PAY.OT,HOURS.REG,HOURS.OT

    READ # FILE.NO; NAME$,ADDRESS$,PHONE$,ZIP
```

ELEMENT:

REM statement

SYNOPSIS:

[<line number>] REM [<remark>]

[<line number>] REMARK [<remark>]

DESCRIPTION:

A REM statement is ignored by the compiler and compilation continues with the statement following the next carriage return. The REM statement may be used to document a program. REM statements do not affect the size of program that may be compiled or executed. An unlabeled REM statement may follow any statement on the same line. And the <line number> may occur in a GOTO, GOSUB or ON statement.

EXAMPLES:

```
10  REM THIS IS A REMARK

    REMARK THIS IS ALSO A REMARK

    LET X = 0 REM INITIAL VALUE OF X
```

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved, worldwide

ELEMENT:

reserved word list

SYNOPSIS:

<letter> { <letter> } [$]

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved, worldwide

DESCRIPTION:

The following words are reserved by BASIC-E and may not be used as <identifiers>:

ABS	AND	ASC	ATN	CHR$
CLOSE	COS	COSH	DATA	DEF
DIM	ELSE	END	EQ	EXP
FILE	FOR	FRE	GE	GO
GOSUB	GOTO	GT	IF	INP
INPUT	INT	LE	LEFT$	LEN
LET	LOG	LT	MID$	NE
NEXT	NOT	ON	OR	OUT
POS	PRINT	RANDOMIZE	READ	REM
RESTORE	RETURN	RIGHT$	RND	SGN
SIN	SINH	SQR	STEP	STOP
STR$	SUB	TAB	TAN	THEN
TO	VAL			

Reserved words must be preceeded and followed by either a special character or a space. Spaces may not be embedded within reserved words. Unless compiler toggle D is set, lowercase letters are converted to uppercase prior to checking to see if an <identifier> is a reserved word.

ELEMENT:

RESTORE statement

SYNOPSIS:

[<line number>] RESTORE

© 1977 IMSAI MFG. CORP.
San Leandro, CA. Made in U.S.A.
All rights reserved, worldwide

DESCRIPTION:

A RESTORE statement repositions the pointer into the data area so that the next value read with a READ statement will be the first item in the first DATA statement. The effect of a RESTORE statement is to allow rereading the DATA statements.

EXAMPLES:

```
    RESTORE

10  RESTORE
```

ELEMENT:

RETURN statement

SYNOPSIS:

[<line number>] RETURN

DESCRIPTION:

Control is returned from a subroutine to the calling routine. The return address is maintained on the top of the run-time monitor stack. No check is made to insure that the RETURN follows a GOSUB statement.

EXAMPLES:

```
130 RETURN

    RETURN
```

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved, worldwide

ELEMENT:

RIGHT$ predefined function

SYNOPSIS:

RIGHT$ (<expression> , <expression>)

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved, worldwide

DESCRIPTION:

The RIGHT$ function returns the n rightmost characters of the first <exprssion>. The value of n is equal to the integer portion of the second <expression>. If n is negative an error occurs; if n is greater than the length of the first <expression> then the entire <expression> is returned. The first argument must produce a string and the second must produce a floating point number.

EXAMPLES:

```
RIGHT$(X$,1)

RIGHT$(NAME$,LNG.LAST)
```

ELEMENT:

RND predefined function

SYNOPSIS:

RND

DESCRIPTION:

The RND function generates a uniformly distributed random number between 0 and 1.

EXAMPLE:

RND

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved worldwide

ELEMENT:

SGN predefined function

SYNOPSIS:

SGN (<expression>)

DESCRIPTION:

The SGN function returns 1 if the value of the <expression> is greater than 0, -1 if the value is less than 0 and 0 if the value of the <expression> is 0. The argument must evaluate to a floating point number.

EXAMPLES:

```
SGN(X)

SGN(A - B + C)
```

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved, worldwide

ELEMENT:

SIN predefined function

SYNOPSIS:

SIN (<expression>)

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved, worldwide

DESCRIPTION:

SIN is a predefined function which returns the sine of the <expression>. The argument must evaluate to a floating point number in radians.

A floating point overflow occurs if the absolute value of the <expression> is greater than two raised to the 24th power times pi.

EXAMPLES:

```
X = SIN(Y)

SIN(A - B/C)
```

ELEMENT:

SINH predefined function

SYNOPSIS:

SINH (<expression>)

DESCRIPTION:

SINH is a function which returns the hyperbolic sine of the <expression>. The argument must evaluate to a floating point number.

EXAMPLES:

SINH(Y)

SINH(B<C)

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved, worldwide

ELEMENT:

special characters

DESCRIPTION:

The following special characters are used by BASIC-E:

↑	circumflex
(	open parenthesis
)	closed parenthesis
*	asterisk
+	plus
-	minus
/	slant
:	colon
;	semicolon
<	less-than
>	greater-than
=	equal
#	number-sign
,	comma
CR	carriage return
\	backslant

Any special character in the ASCII character set may appear in a string. Special characters other than those listed above, if they appear outside a string, will generate an IC error.

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved, worldwide

ELEMENT:

<statment>

SYNOPSIS:

[<line number>] <statement list> <cr>

[<line number>] IF statement <cr>

[<line number>] DIM statement <cr>

[<line number>] DEF statement <cr>

[<line number>] END statement <cr>

DESCRIPTION:

All BASIC-E statments are terminated by a carriage return (<cr>).

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved, worldwide

ELEMENT:

<statment list>

SYNOPSIS:

<simple statement> {: <simple statement> }

where a <simple statement> is one of the following:

FOR statement
NEXT statement
FILE statement
CLOSE statement
GOSUB statement
GOTO statment
INPUT statement
LET statement
ON statement
PRINT statement
READ statement
RESTORE statement
RETURN statement
RANDOMIZE statement
OUT statement
STOP statement
<empty> statement

DESCRIPTION:

A <statement list> allows more than one <statement> to occur on a single line.

EXAMPLES:

```
LET I = 0 : LET J = 0 : LET K = 0

X = Y+Z/W : RETURN

:::::: PRINT "THIS IS OK TOO"
```

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved. worldwide

ELEMENT:

STR$ predefined function

SYNOPSIS:

STR$ (<expression>)

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved, worldwide

DESCRIPTION:

The STR$ function returns the ASCII string which represents the value of the <expression>. The argument must evaluate to a floating point number.

EXAMPLES:

STR$(X)

STR$(3.141617)

ELEMENT:

<subscript list>

SYNOPSIS:

<expression> {, <expression> }

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved, worldwide

DESCRIPTION:

A <subscript list> may be used as part of a <DIM statement> to specify the number of dimensions and extent of each dimension of the array being declared or as part of a <subscripted variable> to indicate which element of an array is being referenced.

There may be any number of expressions but each must evaluate to a floating point number. A <subscript list> as part of a DIM statement may not contain a reference to the array being dimensioned.

EXAMPLES:

```
X(10,20,20)

Y$(I,J)

COST(AMT(I),PRICE(I))
```

ELEMENT:

SQR (<expression>)

DESCRIPTION:

SQR returns the square root of the absolute value of the <expression>. The argument must evaluate to a floating point number.

EXAMPLES:

```
SQR (Y)

SQR(X↑2 + Y↑2)
```

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved, worldwide

ELEMENT:

TAB predefined function

SYNOPSIS:

TAB (<expression>)

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved, worldwide

DESCRIPTION:

The TAB function positions the output buffer pointer to the position specified by the integer value of the <expression> rounded to the nearest integer modulo 73. If the value of the rounded expression is less than or equal to the current print position, the print buffer is dumped and the buffer pointer is set as described above.

The TAB function may occur only in PRINT statements.

EXAMPLES:

```
TAB(10)

TAB(I + 1)
```

ELEMENT:

STOP statement

SYNOPSIS:

[<line number>] STOP

DESCRIPTION:

Upon encountering a <STOP statement> program execution terminates and all open files are closed. The print buffer is emptied and control returns to the host system. Any number of STOP statements may appear in a program.

A STOP statement is appended to all programs by the compiler.

EXAMPLES:

```
10  STOP

    STOP
```

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved, worldwide

ELEMENT:

TAN predefined function

SYNOPSIS:

TAN (<expression>)

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved. worldwide

DESCRIPTION:

TAN is a function which returns the tangent of the expression. The argument must be in radians.

An error occurs if the <expression> is a multiple of pi/2 radians.

EXAMPLES:

```
10  TAN(A)

    TAN(X - 3*COS(Y))
```

ELEMENT:

VAL predefined function

SYNOPSIS:

VAL (<expression>)

DESCRIPTION:

The VAL function converts the number in ASCII passed as a parameter into a floating point number. The <expression> must evaluate to a string.

Conversion continues until a character is encountered that is not part of a valid number or until the end of the string is encountered.

EXAMPLES:

```
VAL(A$)

VAL("3.789" + "E-07" + "THIS IS IGNORED")
```

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved worldwide

ELEMENT:

<variable>

SYNOPSIS:

<identifier> [(<subscript list>)]

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved, worldwide

DESCRIPTION:

A <variable> in BASIC-E may either represent a floating point number or a string depending on the type of the <identifier>. Subscripted variables must appear in a DIM statement before being used as a <variable>.

EXAMPLES:

```
X

Y$(3,10)

ABS.AMT(X(I),Y(I),S(I-1))
```

III. OPERATING INSTRUCTIONS FOR BASIC-E

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved, worldwide

The BASIC-E programs are written to operate with the CP/M Floppy Disk Operating System (4). Operation with a different system will require modification to the input/output routines in the compiler and run-time monitor. Execution of a program using BASIC-E consists of three steps. First the source program must be created on disk. Next the program is compiled by executing the BASIC-E compiler with the name of the source program provided as a parameter. Finally the intermediate (INT) file created by the compiler may be interpreted by executing the run-time monitor, again using the the source program name as a parameter.

Creation of the source program will normally be accomplished using CP/M's text editor, and must have a file type BAS. The BASIC-E statements are free form with the restriction that when a statement is not completed on a single line, a continuation character (\) must be the last character on the line. Spaces may precede statements and any number of spaces may appear wherever one space is permitted. Line numbers need only be used on statements to which control is passed. The line numbers do not have to be in ascending order. Using identifiers longer than two characters and indenting statements to enhance readability does

not affect the size of the object file created by the compiler.

The first statement of a source program may be used to specify certain compiler options. If present, this statement must begin with a dollar sign ($) in column one and be followed by the letter or letters indicating the options which are desired. The letters may be separated by any number of blanks. Invalid letters or characters are ignored. Appendix D lists valid compiler options, and their initial settings. Toggle A is used for compiler debugging. Toggle B supresses listing of the source program except for statements with errors. Toggle C compiles the program but does not create a INT file. Normally the BASIC-E compiler converts all letters appearing in identifiers or reserved words to uppercase. If toggle D is set this conversion is not performed. Letters appearing in strings are never converted to uppercase. Toggle E causes code to be generated by the compiler so that, upon detection of a run-time error, the source statement line which was being executed at the time the error occured is listed along with the error message.

The BASIC-E compiler is invoked as follows:

BASIC <program name>

The compiler begins execution by opening the source file specified as a parameter and compiles each BASIC-E statement producing an object file in the BASIC-E machine language with the same name as the source program but of type "INT".

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved, worldwide

The source program may be listed on the output device with any error messages following each line of the program. Appendix A lists the compiler error messages. Errors SO, TO and VO indicate storage used by the compiler has been exceeded. These errors require recompilation of the BASIC-E compiler with more space allocated to the particular vector involved. Figures 1 and 2 show sample program listings. If no errors occur during compilation, the object file may be executed by the run time monitor by typing the command:

RUN <program name>

The run-time monitor consists of two programs. The first program initializes the floating point package and then reads the intermediate language file specified as a parameter. As the INT file is read, the floating point constants, BASIC-E machine code, and the data area are built. At this time branch addresses and references to floating point constants and the program reference table (PRT) are relocated to reflect actual machine addresses. Control is then passed to the interpreter for execution of the BASIC-E machine language. The program which builds the machine resides in the free storage area and is overwritten as space is allocated by the interpreter. Appendix B lists run-time error messages. Figure 3 outlines the structure of the BASIC-E machine.

The BASIC-E machine is divided into a static and varying area. The static section consists of the BASIC-E Run Time Monitor, constants, machine code and data statements.

© 1977 · IMSAI MFG. CORP
San Le·andro, CA. Made in U.S.A.
All rights re·served, worldwide

```
REMARK  PROGRAM TO COMPUTE THE FIRST N
REMARK  FIBONACCI NUMBERS

    PRINT "THIS PROGRAM COMPUTES THE FIRST N"
    PRINT "FIBONACCI NUMBERS"
    PRINT "AN INPUT OF 0 TERMINATES THE PROGRAM"

FOR I = 1 TO 1 STEP 0  REMARK DO THIS FOREVER
    INPUT "ENTER THE VALUE OF N"; N
    IF N = 0 THEN \
        PRINT "PROGRAM TERMINATES" :\
        STOP
    IF N < 0 THEN \
        PRINT "N MUST BE POSITIVE. "; :\
        PRINT "PLEASE REENTER" \
    ELSE \
        GOSUB 300 REMARK CALCULATE AND PRINT RESULTS
NEXT I

300     REMARK SUBROUTINE TO CALCULATE FIB NUMBERS
    F1 = 1 : F2 = 1  REMARK INITIAL VALUES
    NUM = F1

REMARK HANDLE FIRST TWO NUMBERS (IF REQ) AS
REMARK SPECIAL CASES

    FOR J = 1 TO 2.
        GOSUB 400
        IF N = 0 THEN \
            RETURN
    NEXT J

REMARK  HANDLE REMAINING NUMBERS

    FOR J = 1 TO 1 STEP 0
        NUM = F1 + F2
        GOSUB 400
        F2 = F1
        F1 = NUM
        IF N = 0 THEN \
            RETURN
    NEXT J
    RETURN

400     REMARK PRINT NEXT NUMBER AND
        REMARK DECREMENT N
    PRINT NUM, REMARK 5 TO A LINE
    N = N - 1
    RETURN
END
```

Figure 1

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved, worldwide

© 1977 IMSAI MFG. CORP.
San Leandro, CA. Made in U.S.A.
All rights reserved, worldwide

```
REMARK  PROGRAM BUILDS A FILE OF MAILING LABELS
REMARK  FROM A FILE CONTAINING 100 BYTE RECORDS
REMARK  WHICH CONTAIN NAME AND ADDRESS INFORMATION

    INPUT "NAME OF SOURCE FILE";SOURCE.FILE$
    INPUT "NAME OF LABEL FILE";LABLE.FILE$
    IF END # 1 THEN 100
    FILE SOURCE.FILE$(100), LABLE.FILE
REMARK LABLE.FILE IS NOT BLOCKED

    FOR INDEX = 1 TO 1 STEP 0  REM UNTIL END OF FILE
        READ # 1; FIRST$, LAST$, STREET$, CITY$, STATE$,ZIP

        REMARK LINES ARE TRUNCATED AT 60 CHARACTERS

        LINE1$ = LEFT$(FIRST$ + " " + LAST$,60)
        LINE2$ = LEFT$(STREET$,60)

        REMARK INSURE ZIP NOT TRUNCATED

        LINE3$ = LEFT$(CITY$ + ", " + STATE$,54)
        LINE3$ = LINE3$ + " " + STR$(ZIP)

        PRINT # 2; LINE1$
        PRINT # 2; LINE2$
        PRINT # 2; LINE3$
    NEXT INDEX

100     PRINT "JOB COMPLETE"
        STOP
    END
```

Figure 2

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved, worldwide

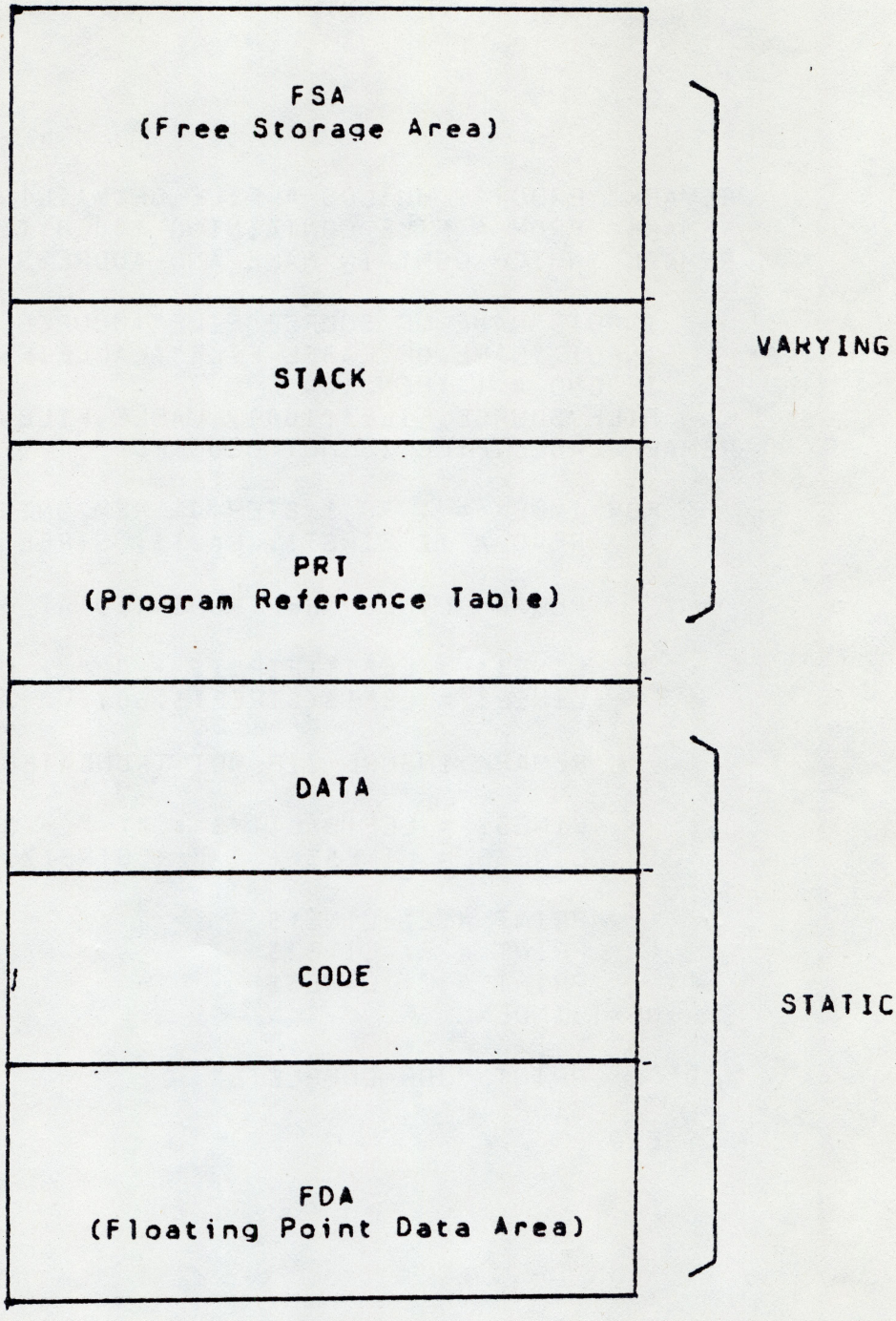


Figure 3

Structure of the NBASIC Machine

The varying area, which includes all the remaining memory, stores program variables, the stack and the free storage area. The stack is a circular queue and therefore it is not possible to overflow the stack. However, if the stack wraps around onto itself the results could be meaningless. The size of the stack is initially set at 12 but may be changed by recompiling the interpreter. The free storage area is used to dynamically allocate arrays and strings. Storage is freed as soon as it is no longer required by the program.

© 1977 IMSAI MFG. CORP
San Le[illegible]ndro, CA. Made in U.S.A.
All rights re[illegible]served, worldwide

All requests for input from the console are prompted with a question mark. If more information is entered than was requested, or if insufficient information is entered, a warning occurs and the entire line of data must be entered again. A program may be terminated by typing a control-Z followed by a carriage return in response to an input request.

Disk files may be read, written or updated by the BASIC-E programmer using both sequential and random access. There are two basic types of files: those with no declared record size, which are referred to as unblocked, and those with user-specified record size. The latter are blocked files. Blocked files may be processed either sequentially or randomly, while unblocked files must be accessed sequentially. All data in files is stored as ASCII characters. Either a comma or carriage return denotes the end of a field. Blanks are ignored between fields.

In unblocked files there is no concept of a record as such. Each reference to the file either reads from or writes to the next field. At the end of each <write statement> a carriage return and a linefeed are written to the file. This provides compatability with the operating system utilities such as TYPE and also allows files to be created by the text editor.

Blocked files consist of a series of fixed length records. The user specifies the logical record length with the <file statement>. An error occurs if a linefeed is encountered during a read from a blocked file or if the current record being built exceeds the block size during a write. At the end of a <write statement> any remaining area in the record is filled with blanks, and then a carriage return and linefeed are added.

© 1977 IMSAI MFG. CORP.
San Leandro, CA. Made in U.S.A.
All rights reserved, worldwide

APPENDIX A - COMPILER ERROR MESSAGES

© 1977 · IMSAI MFG. CORP.
San Leandro, CA. Made in U.S.A.
All rights reserved, worldwide

CE	could not close file.
DE	Disk error.
DF	Could not create INT file; disk or directory is full.
DL	Duplicate labels or syncronization error.
DP	Identifier in DIM statement previously defined.
FC	Identifier in FILE statement previously defined.
FD	Predefined function name previously defined.
FI	FOR loop index is not a simple floating point variable.
FN	Incorrect number of parameters in function reference.
FP	Invalid parameter type in function reference.
FU	Function is undefined.
IC	Invalid character in BASIC statement.
IE	Expression in IF statement is not of type floating-point.
IS	Subscripted variable not previously dimensioned.
IU	Array name used as simple variable.
MF	Expression is of type string where only floating point is allowed.
MM	Expression contains string and floating point variables in mixed mode expression.
NI	Identifier following NEXT does not match FOR statement index.
NP	No applicable production exists.
NS	No BAS file found.

NU	NEXT statement without corresponding FOR statement.
SN	Incorrect number of subscripts.
SO	Compiler stack overflow.
TO	Symbol table overflow.
UL	Undefined label.
VO	VARC overflow.

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved, worldwide

APPENDIX B - RUN-TIME MONITOR ERROR MESSAGES

AC	Null string passed as parameter to ASC function.
CE	Error closing a file.
DR	Disk read error (reading unwritten data in random access).
DW	Error writing to a file.
DZ	Division by zero.
EF	Eof on disk file.; no action specified.
ER	Exceeded record size on block file.
II	Invalid input from the console.
IR	Invalid record number in random access.
FU	Accessing an unopened file.
ME	Error attempting to create a file.
MF	File identifier too large or zero.
NE	Attempt to raise a number to a negative power.
NI	No INT file found in directory.
OD	Attempt to read past end of data area.
OE	Error attempting to open a file.
OI	Index in ON statement out of bounds.
RE	Attempt to read past end of record on blocked file.
RU	Unblocked file used with random access.
SB	Array subscript out of bounds.
SL	String length exceeds 255.
SS	Second parameter of MID$ is negative.
TZ	Attempt to evaluate tangent of pi over two.

© 1977 IMSAI MFG. CORP
San Le[illegible]ndro, CA. Made in U.S.A.
All rights [illegible]served, worldwide

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved, worldwide

APPENDIX C - BASIC-E GRAMMAR

```
 1   <program> ::= <line number> <statement> <cr>

 2   <line number> ::= <number>
 3                   ¦

 4   <statement> ::= <statement list>
 5                 ¦ <if statement>
 6                 ¦ <end statement>
 7                 ¦ <dimension statement>
 8                 ¦ <define statement>

 9   <statement list> ::= <simple statement>
10                     ¦ <statement list> :
10                       <simple statement>

11   <simple statement> ::= <let statement>
12                        ¦ <assignment>
13                        ¦ <for statement>
14                        ¦ <next statement>
15                        ¦ <file statement>
16                        ¦ <close statement>
17                        ¦ <read statement>
18                        ¦ <print statement>
19                        ¦ <goto statement>
20                        ¦ <gosub statement>
21                        ¦ <input statement>
22                        ¦ <stop statement>
23                        ¦ <return statement>
24                        ¦ <on statement>
25                        ¦ <restore statement>
26                        ¦ <randomize statement>
27                        ¦ <out statement>
28                        ¦

29   <let statement> ::= let <assignment>

30   <assignment> ::= <assign head> <expression>

31   <assign head> ::= <variable> =

32   <expression> ::= <logical factor>
33                  ¦ <expression> <or> <logical factor>

34   <or> ::= or
35          ¦ xor

36   <logical factor> ::= <logical secondary>
37                      ¦ <logical factor> and
```

```
37                          <logical secondary>.

38   <logical secondary> ::= <logical primary>
39                         ¦ not <logical primary>

40   <logical primary> ::= <arithmetic expression>
41                        ¦ <arithmetic expression>
41                          <relation>
41                          <arithmetic expression>

42   <arithmetic expression> ::= <term>
43                             ¦ <arithmetic expression> +
43                               <term>
44                             ¦ <arithmetic expression> -
44                               <term>
45                             ¦ + <term>
46                             ¦ - <term>

47   <term> ::= <primary>
48            ¦ <term> * <primary>
49            ¦ <term> / <primary>

50   <primary> ::= <element>
51               ¦ <primary> ** <element>

52   <element> ::= <variable>
53               ¦ <constant>
54               ¦ <function call>
55               ¦ ( <expression> )

56   <variable> ::= <identifier>
57                 ¦ <subscript head> <expression> )

58   <subscript head> ::= <identifier> (
59                       ¦ <subscript head> <expression> ,

60   <function call> ::= <function heading> <expression> )
61                      ¦ <function name>

62   <function heading> ::= <function name> (
63                         ¦ <function heading> <expression>
63                           ,

64   <function name> ::= <userdefined name>
65                      ¦ <predefined name>

66   <constant> ::= <number>
67                 ¦ <string>

68   <relation> ::= =
69                 ¦ > =
70                 ¦ ge
71                 ¦ < =
72                 ¦ le
73                 ¦ >
```

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved, worldwide

```
74                  ¦ <
75                  ¦ < >
76                  ¦ ne

77   <for statement> ::= <for head> to <expression>
77                          <step clause>

78   <for head> ::= <for> <assignment>

79   <for> ::= for

80   <step clause> ::= step <expression>
81                     ¦

82   <if statement> ::= <if group>
83                      ¦ <if else group> <statement list>
84                      ¦ if end # <expression> then <number>

85   <if group> ::= <if head> <statement list>
86                  ¦ <if head> <number>

87   <if else group> ::= <if head> <statement list> else

88   <if head> ::= if <expression> then

89   <define statement> ::= <ud function name>
89                             <dummy arg list> = <expression>

90   <ud function name> ::= def <userdefined name>

91   <dummy arg list> ::= <dummy arg head> <identifier> )
92                         ¦

93   <dummy arg head> ::= (
94                         ¦ <dummy arg head> <identifier> ,

95   <file statement> ::= <file head> <file decleration>

96   <file head> ::= file
97                   ¦ <file head> <file decleration> ,

98   <file decleration> ::= <identifier> <file rec size>

99   <file rec size> ::= ( <expression> )
100                        ¦

101  <dimension statement> ::= dim
101                               <dimension variable list>

102  <dimension variable list> ::= <dimension variable>
103                                 ¦
103                                  <dimension variable list>
103                                   , <dimension variable>

104  <dimension variable> ::= <dim var head> <expression> )
```

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved, worldwide

```
105   <dim var head> ::= <identifier> (
106                    ¦ <dim var head> <expression> ,

107   <close statement> ::= close <close list>

108   <close list> ::= <expression>
109                  ¦ <close list> , <expression>

110   <read statement> ::= read <file option> <read list>
111                      ¦ read <read list>

112   <input statement> ::= input <prompt option>
112                           <read list>

113   <prompt option> ::= <constant> ;
114                     ¦

115   <read list> ::= <variable>
116                 ¦ <read list> , <variable>
117                 ¦

118   <print statement> ::= print <print list> <print end>
119                       ¦ print <file option> <file list>

120   <print list> ::= <expression>
121                  ¦ <print list> <print delim>
121                    <expression>
122                  ¦

123   <file list> ::= <expression>
124                 ¦ <file list> , <expression>

125   <print end> ::= <print delim>
126                 ¦

127   <file option> ::= # <expression> ;
128                   ¦ # <expression> , <expression> ;

129   <print delim> ::= ;
130                   ¦ ,

131   <goto statement> ::= <goto> <number>

132   <on statement> ::= <on goto> <label list>
133                    ¦ <on gosub> <label list>

134   <on goto> ::= on <expression> <goto>

135   <on gosub> ::= on <expression> <gosub>

136   <label list> ::= <number>
137                  ¦ <label list> , <number>

138   <gosub statement> ::= <gosub> <number>
```

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved, worldwide

```
139   <goto> ::= goto
140            ¦ go to

141   <gosub> ::= gosub
142             ¦ go sub

143   <next statement> ::= <next head> <identifier>
144                      ¦ next

145   <next head> ::= next
146                 ¦ <next head> <identifier> ,

147   <out statement> ::= out <expression> , <expression>

148   <return statement> ::= return

149   <stop statement> ::= stop

150   <end statement> ::= end

151   <restore statement> ::= restore

152   <randomize statement> ::= randomize
```

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved, worldwide

APPENDIX D - COMPILER OPTIONS

Option	Description	Default
A	List productions (for compiler debugging).	OFF
B	List only source statements with errors.	OFF
C	Do not create INT file; syntax check only.	OFF
D	Convert lowercase to uppercase.	ON
E	Generate line number code.	OFF

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved, worldwide

BIBLIOGRAPHY

1. Draft Proposed American National Standard Programming Language Minimal BASIC. X3J2/76-01 76-01-01. Technical Committee X3J2-BASIC American National Standards Committee X3 - Computers and Information Processing.

2. Worth, Thomas. BASIC For Everyone. Englewood Cliffs: Prentice Hall, Inc., 1976.

3. Albrecht, Fobert L., Fenkel, LeRoy and Brown, Jerry. BASIC. New York: John Wiley & Sons, Inc., 1973.

4. Kildall, G. A. An Introduction to CP/M Features and Facilities. Digital Research, 1976.

© 1977 IMSAI MFG. CORP
San Leandro, CA. Made in U.S.A.
All rights reserved, worldwide